# Revisiting the Role of Feature Engineering for Compound Type Identification in Sanskrit

**Jivnesh Sandhan[1], Amrith Krishna[2], Pawan Goyal[2] and Laxmidhar Behera[1]**
[1] Department of Electrical Engineering
Indian Institute of Technology, Kanpur, UP, India
[2]Department of Computer Science
Indian Institute of Technology, Kharagpur, WB, India
jivnesh@iitk.ac.in

## Abstract

We propose an automated approach for semantic class identification of compounds in Sanskrit. It is essential to extract semantic information hidden in compounds for improving overall downstream Natural Language Processing (NLP) applications such as information extraction, question answering, machine translation, and many more. In this work, we systematically investigate the following research question: Can recent advances in neural network outperform traditional hand engineered feature based methods on the semantic level multi-class compound classification task for Sanskrit? Contrary to the previous methods, our method does not require feature engineering. For well-organized analysis, we categorize neural systems based on Multi-Layer Perceptron (MLP), Convolution Neural Network (CNN) and Long Short Term Memory (LSTM) architecture and feed input to the system from one of the possible levels, namely, word level, sub-word level, and character level. Our best system with LSTM architecture and FastText embedding with end-to-end training has shown promising results in terms of F-score (0.73) compared to the state of the art method based on feature engineering (0.74) and outperformed in terms of accuracy (77.68%).

## 1 Introduction

The landscape of Natural Language Processing has significantly shifted towards the realm of Deep Learning and Artificial Neural Networks. With the benefit of hindsight, the title for the seminal work on a neural pipeline for NLP from Collobert et al. (2011), "Natural Language Processing (Almost) from Scratch", seems prophetic. Neural networks have demonstrated promising results in a wide variety of problems like sentiment analysis (Tai et al., 2015), information extraction (Nguyen et al., 2009), text classification (Kim, 2014), machine translation (Bastings et al., 2017) among others. Many of such models in fact have become part and parcel of a standard NLP pipeline for data processing, especially for the resource-rich languages such as English (Tenney et al., 2019).

There have been academic debates over the philosophical implications of the use of such statistical black box approaches in Computational Linguistics, especially towards the trade-off between performance and interpretability as also summarised in Krishna et al. (2018b). However, in this work, we focus more on the pragmatic side of using such approaches for low resource languages like Sanskrit. Deep Learning models demand a humongous amount of data to train a model effectively. Additionally, it is challenging and often tricky to incorporate available linguistic knowledge into these neural architectures (Strubell et al., 2018). Summarily, we can say that a standard off the shelf neural model relies mostly on its capacity to learn distributional information from the large datasets provided as input during training. In this pretext, we revisit the problem of compound type identification in Sanskrit (Krishna et al., 2016) and experiment with various neural architectures for solving the task.

The process of compounding and the nature of compositionality of the compounds are well studied in the field of NLP. Given that compounding is a productive process of word-formation in languages, this is of much interest in the area of word-level semantics in NLP. There are various aspects involved in the compound analysis. These include productivity and recursiveness of the words involved in the process, presence of implicit relations between the components, and finally, the analysis of a compound relies on its pragmatic or contextual features (Kim and Baldwin, 2005). Recently, there has been a concerted effort in studying the nature of compositionality in compounds by leveraging on distributional word-representations or word embeddings and then learning function approximators to predict the nature of compositionality of such words (Mitchell and Lapata, 2010; Cordeiro et al., 2016; Salehi et al., 2015; Jana et al., 2019). In Sanskrit, Krishna et al. (2016) have proposed a framework for semantic type classification of compounds in Sanskrit. They proposed a multi-class classifier using Random Forests (Geurts et al., 2006; Pedregosa et al., 2011), where they classified a given compound into one of the four coarse level compound classes, namely, *Avyayībhāva, Tatpuruṣa, Bahuvrīhi* and *Dvandva*. They have used an elaborate feature set, which summarily consists of rules from the grammar treatise *Aṣṭādhyāyī* pertaining to compounding, semantic relations between the compound components from a lexical database *Amarakoṣa* and distributional subword patterns from the data using Adaptor Grammar (Johnson et al., 2007). Inspired from the recent advances in using neural models for compound analysis in NLP, we revisit the task of compound class identification and validate the efficacy of such models under the low-resource setting like that of Sanskrit.

In this work, we experiment with multiple deep learning models for compound type classification. Our extensive experiments include standard neural models comprising of Multi-Layer Perceptrons (MLP), Convolution Neural Networks (CNN) (Zhang et al., 2015) and Recurrent models such as Long Short Term Memory (LSTM) configurations. Unlike the feature-rich representation of Krishna et al. (2016), we rely on various word embedding approaches, which include character level, sub-word level, and word-level embedding approaches. Using end-to-end training, the pretrained embeddings are fine tuned for making them task specific embeddings. So all the architectures are integrated with end-to-end training (Kim, 2014). The best system of ours, an end-to-end LSTM architecture initialised with fasttext embeddings has shown promising results in terms of F-score (0.73) compared to the state of the art classifier from Krishna et al. (2016) (0.74) and outperformed it in terms of accuracy (77.68%). Summarily, we find that the models we experimented with, report competitive results with the current state of the art model for compound type identification. We achieve the same without making use of any feature engineering or domain expertise. We release the codebase for all our models experimented with at `https://github.com/Jivnesh/ISCLS-19`.

## 2 Compound Classification Task in Sanskrit

In this work, we address the challenge of semantic type identification of compounds in Sanskrit. This is generally treated as a word-level semantic task in NLP (Rink and Harabagiu, 2010; Hashimoto et al., 2014; Santos et al., 2015). We treat the task as a supervised multiclass classification problem. Here, similar to Krishna et al. (2016), we expect the users to provide a compound in its component-wise segmented form as input to the model. But our model relies on distributed representations or embeddings of the input as features, instead of the linguistically involved feature set proposed in Krishna et al. (2016).

Approaches for compound analysis have been of great interest in NLP for multiple languages including English, Italian, Dutch and German (Séaghdha and Copestake, 2013; Tratz and Hovy, 2010; Kim and Baldwin, 2005; Girju et al., 2005; Verhoeven et al., 2014a). These methods primarily rely on lexical networks, distributional information (Séaghdha and Copestake, 2013) or a combination of both lexical and distributional information (Nastase et al., 2006). In Sanskrit, Krishna et al. (2016) proposed a similar statistical approach which combined lexical and distributional information by using information from the lexical network *Amarakoṣa* (Nair and

Kulkarni, 2010) and variable length n-grams learned from data using Adaptor grammar (Johnson et al., 2007). Here, the authors also adopted rules from *Aṣṭādhyāyī* as potentially discriminative features for compound type identification (Kulkarni and Kumar, 2013). While this model has shown to be effective for the task, it nevertheless is a linguistically involved model. Recently, Dima and Hinrichs (2015), Cordeiro et al. (2016) and Ponkiya et al. (2016) have shown that use of word embedding as the sole features can produce models with competitive results as compared to other feature-rich models. Inspired from these observations, we attempt to build similar models which use only embeddings as features for the compound type identification task.

Compounds in Sanskrit can be categorized into 30 possible classes based on how granular categorizations one would like to have (Lowe, 2015). There are slightly altered set of categorizations considered by Gillon (2009), Olsen (2000), Bisetto and Scalise (2005) and Tubb and Boose (2007). Semantically *Aṣṭādhyāyī* categorizes the Sanskrit compounds into four major semantic classes, namely, *Avyayībhāva, Tatpuruṣa, Bahuvrīhi* and *Dvandva* (Kumar et al., 2010). Similar to prior computational approaches in Sanskrit compounding (Krishna et al., 2016; Kumar et al., 2010), we follow this four class coarse level categorization of the semantic classes in compounds. Compounding in Sanskrit is extremely productive, or rather recursive, resulting in compound words with multiple components (Lowe, 2015). Further, it is observed that compounding of a pair of components may result in compounds of different semantic classes. *Avyayībhāva* and *Tatpuruṣa* may likely be confusing due to particular sub-category of *Tatpuruṣa* if the first component is an *avyaya*. For example, *upa jīvataḥ* has the first component as *avyaya* which is strong characteristic of *Avyayībhāva*. However, this compound belongs to *Tatpuruṣa* class. Likewise, a negation *avyaya* in the first component can create confusion between *Tatpuruṣa* and *Bahuvrīhi* classes. The instances mentioned above reveal the difficulties associated with distinguishing the semantic classes of a compound.

## 3 System Description

While the compounds in Sanskrit can consist of multiple components, we restrict our problem to that of compounds with two components only. Thus, given the two components of the compound, we treat this as a classification problem. For the task, we use neural models, which can be categorized based on the architectural point of view, namely, Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) based classifier, among others.

These networks typically require a feature representation of the input (in our case, the two components of the compound word), and learn to classify into one of the possible compound categories. We again utilize multiple possibilities of input feature representation. For instance, consider *svamānasam*, which is a *Tatpuruṣa* compound. We can break this compound in three possible ways: 1) word level: *sva mānasam* 2) subword level: *sva mā nas am* (subword level segmentation is based on segmentation learned by Byte Pair Encoding (BPE) (Sennrich et al., 2016) from corpus data). 3) Character level: *s v a m ā n a s a m*.

We learned word embeddings of these components of the compound from our Sanskrit corpus (Section 4.1). Word embeddings map a word $x$ from a vocabulary $V$ to a real-valued vector $\vec{x}$ of dimensionality $D$ in a feature space (Schnabel et al., 2015). The idea based on distributional hypothesis (Harris, 1954), and the learning objective attempts to put similar words closer in the vector space.We used FastText for learning word-level embedding, BPE along with Word2Vec (w2v) (Mikolov et al., 2013) and Glove (Pennington et al., 2014) for learning subword level embedding, and character level embedding learned using CharCNN (Zhang et al., 2015). Note that we learned embeddings for the individual components, and finally concatenated vectors corresponding to each component and fed as input to the classifier.

We also integrated our system with task-specific end-to-end training for text classification (Kim, 2014). This approach facilitates pre-trained initialized vector to be updated during the task-specific training process. Performance of the classifier, with and without end-to-end

training, is reported in Appendix I. In all the architectures, *relu* activation function for dense layer, softmax cross entropy loss function and *adam* optimizer are used.

### 3.1 MLP based classifier

Multi-layer Perceptron in supervised learning problem consists of an input layer to receive input, output layer to make a decision and multiple hidden layers in between them. Training involves learning the parameters of the model using backpropagation. As discussed earlier, We experiment with feeding input in two levels, namely, word level (FastText and FastText*) and subword level (W2V and Glove along with BPE). Architectures used for them are reported in Table 1. Next to the embedding layer, a drop-out layer with drop-out rate 0.2 is used to avoid over-fitting (Srivastava et al., 2014).

| Embedding | Layer | units |
|---|---|---|
| w2v [20 x 100] | 1 | 1000 |
| | 2 | 500 |
| | 3 | 100 |
| | 4 | 4 |
| glove [20 x 225] | 1 | 1000 |
| | 2 | 4 |
| FastText [2 x 350] | 1 | 500 |
| | 2 | 4 |
| FastText* [1 x 1400] | 1 | 500 |
| | 2 | 4 |

Table 1: MLP architecture used for different embeddings. [a x b] indicates that there are total 'a' segments of compound and dimension of each segment is 'b'. For instance, for w2v, there are 20 segments (max) to account for the BPE vocabulary of the compound, and each word in the BPE vocabulary is represented using 100 dimensions.

**FastText*:** In this case, as shown in Figure 1, FastText vectors of two components of the compound are concatenated along with element-wise absolute difference and element-wise product between the embedding vector of these two vectors (it is denoted by FastText*). Moreover, the resultant vector is passed to MLP based classifier (Table 1) with no end-to-end training.

The architecture we have used to combine information from the two components is similar to the one used for the Natural Language Inference (NLI) problem in Conneau et al. (2017). The key idea behind their approach was to obtain a unified representation of two sentences, each represented as a vector, similar to Figure 1.

### 3.2 CNN based classifier

CNN has shown outstanding performance in the field of computer vision. The purpose behind adopting CNNs in NLP is to derive position-invariant features (such as phrases, n-grams) using the convolution operation. Max pooling over these features helps to find the essential n-grams and then fully connected hidden layers are employed, similar to MLP, for final predictions. Recently, Kim (2014) has shown the application of CNN for textual data. In our CNN architecture, end-to-end training is integrated into the embedding layer. Next to the embedding layer, drop-out layer with a drop-out rate of 0.2 is used. For different input levels, architecture details are shown in Table 2. Now We will explain CNN used for the character level input.

**CharCNN**: Zhang et al. (2015) used character level information of text as input for a convolutional neural network. The advantage of the model is that by using character level embedding with convolution layers, word-level embedding can be obtained. This model requires fixed-size input of encoded characters where embeddings of each character are initialized with Gaussian

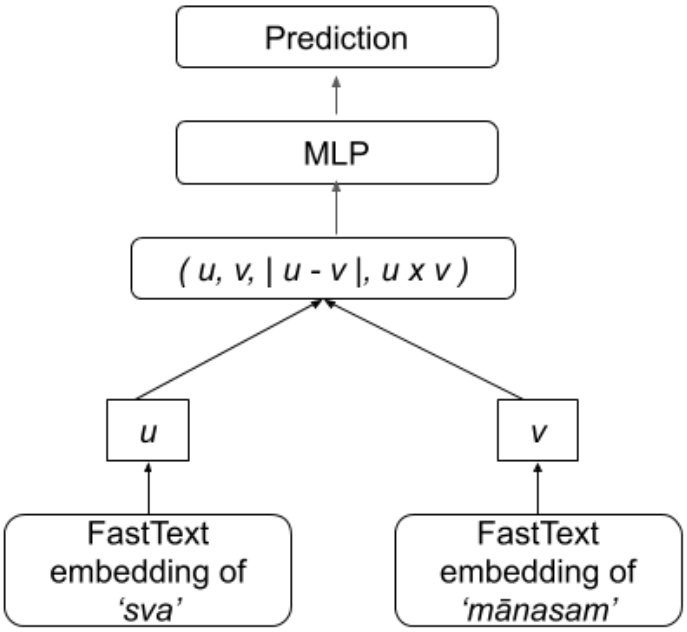



Figure 1: FastText feature augmented with element-wise difference and multiplication of compound's component (Conneau et al., 2017)

distribution with mean 0 and variance 0.05. CharCNN architecture employed for our experiment is mentioned in Table 2.

| Embedding | Layer | Filter | Kernel | Pull |
|---|---|---|---|---|
| CharCNN [25 x 1014] | 1 | 256 | 7 | 3 |
| | 2 | 256 | 7 | 3 |
| | 3 | 256 | 3 | N/A |
| | 4 | 256 | 3 | N/A |
| | 5 | 256 | 3 | N/A |
| | 6 | 256 | 3 | 3 |
| | 7 | 500 | N/A | N/A |
| | 8 | 4 | N/A | N/A |
| w2v [20 x 700] | 1 | 300 | 25 | 4 |
| | 2 | 100 | N/A | N/A |
| | 3 | 4 | N/A | N/A |
| glove [20 x 900] | 1 | 350 | 25 | 4 |
| | 2 | 400 | N/A | N/A |
| | 3 | 4 | N/A | N/A |
| FastText [2 x 350] | 1 | 150 | 25 | 2 |
| | 2 | 4 | N/A | N/A |

Table 2: CNN architecture used for different embeddings. For embedding layer, same convention is used. For charCNN, 25 segments correspond to the max number of characters in the compound, and 1014 dimensional embedding is used for each of these.

### 3.3 LSTM based classifier

The conventional feed-forward neural network treats all input-output pairs independently, which limits the ability to learn patterns in sequential data. RNNs are designed to capture this time dependency where network memorizes the previous input-output interactions in order to predict

the current output. Due to the problem of Vanishing Gradient (Pascanu et al., 2013; Bengio et al., 1994), RNNs can capture only short-term dependencies. To overcome this limitation, LSTM (Hochreiter and Schmidhuber, 1997) is used which employs a gating mechanism to carry forward the long-term dependencies. LSTM has achieved great success in working with sequences of words. In our LSTM architecture, next to embedding layer, drop-out layer with rate 0.2 is used. Embedding layer is integrated with end-to-end training. Architectural details for different input levels are given in Table 3.

| Embedding | Layer | Type | units |
|---|---|---|---|
| w2v [20 x 450] | 1 | LSTM | 450 |
| | 2 | Fully Connected | 400 |
| | 3 | Fully Connected | 4 |
| glove [20 x 900] | 1 | LSTM | 450 |
| | 2 | Fully Connected | 400 |
| | 3 | Fully Connected | 4 |
| FastText [2 x 350] | 1 | LSTM | 100 |
| | 2 | Fully Connected | 4 |

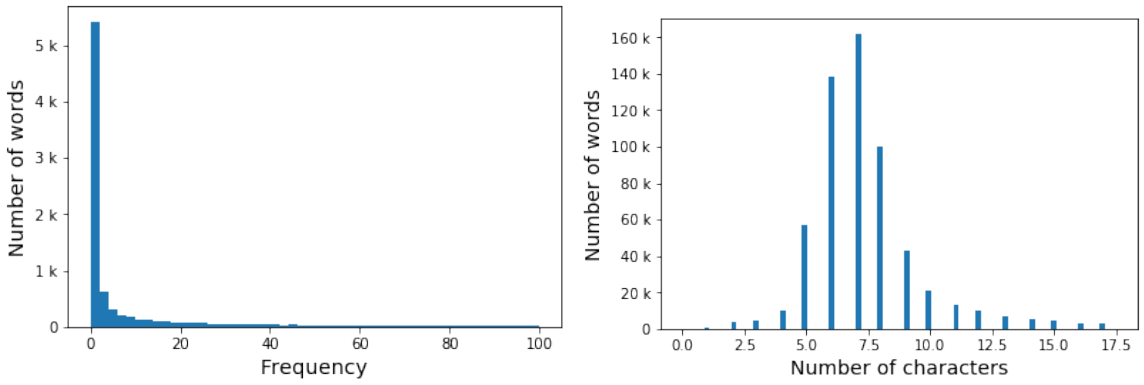
Table 3: LSTM architecture used for different embeddings.

## 4 Experiments

### 4.1 Dataset

Our text corpus contains data from the Digital Corpus of Sanskrit (DCS)[1], as well as scraped data from Wikipedia and Vedabase corpus. The number of words in each corpus are 3.8 M, 0.7 M, and 0.2 M, respectively. DCS and Vedabase are segmented, but the Wikipedia data is unsegmented. We have used this corpus to learn word embedding features. Most of the data in our corpus is in the form of poetry.

Figure 2 presents a few statistics regarding the corpus utilized.




(a) (b)

Figure 2: (a) Histogram plot of frequency of the compounds from the classification dataset in the corpus. 50% of compounds have zero occurrence in the corpus. (b) Distribution of number of characters per word in the corpus.

The labelled dataset for the compound classification task with a segmented pair of components is obtained from the department of Sanskrit studies, UoHyd[2]. These compounds are part of ancient texts, namely, *Bhagavadgītā, Carakasaṃhīta*, etc. We have used the same experimental

[1]http://www.sanskrit-linguistics.org/dcs/
[2]http://sanskrit.uohyd.ac.in/scl/

setting as Krishna et al. (2016) for the classification task. The dataset for the compound classification task has more than 32,000 sandhi splitted compounds with labels. There are four broad classes, namely, *Avyayībhāva, Tatpuruṣa, Bahuvrīhi* and *Dvandva*. More than 75% data points were from *Tatpuruṣa* class, Krishna et al. (2016) down-sampled it to 4,000, which takes it close to the count of the second most highly populated class *Bahuvrīhi*. *Avyayībhāva* class is highly skewed, 5% of the *Bahuvrīhi* class. After down-sampling, number of compounds are 239 in *Avyayībhāva*, 4,271 in *Bahuvrīhi*, 1,176 in *Dvandva*, and 4,266 in *Tatpuruṣa*. Out of 9,952 data-points, 7,957 were kept for training and remaining for testing. We have created development (dev) dataset for hyperparameter tuning, from 20 % stratified sampling of the training data. We have not used test dataset in any part of training or hyperparameter tuning.

### 4.2 Hyperparameter tuning for input representation

Figure 3(e) and 3(f) show the effect of embedding size on the dev set performance. In FastText, accuracy on dev-set saturated at 350, which we used as the default embedding size. Since most of the data is in the form of poetry, the window size is kept larger. As we increase the epoch size, there was a gradual increase in performance (Figure 3(e)). Parameters min-n and max-n were chosen by plotting the distribution of the number of characters in word (Figure 2(b)).

Figure 2(a) shows that more than 50% data sample from the classification task has zero occurrences in the corpus. So this Out of Vocabulary (OOV) issue is handled by applying BPE with vocabulary size 100. Results did not improve by increased vocabulary size of BPE. BPE vocabulary size is chosen as 100, for both glove and w2v features. Embedding for w2v and Glove is calculated for segmented sub-words. Figure 3(b) and 3(c) indicates that by increasing embedding size, there is a gradual increase in F-score on dev dataset for both BPE+W2v and BPE+Glove. So we chose 450 as the embedding size for w2v. For Glove, feature size, epoch size and window size are 450, 70 and 20, respectively.

In CharCNN, the vocabulary size of characters is 60. Apart from the Sanskrit alphabets, there are other eight symbols present in the dataset, which include numbers. The maximum length of characters in the input is 25. Features corresponding to each character is of size 1014, which is initialized from Gaussian distribution with mean 0 and variance 0.05. Filter size, kernel size, and pull size for each layer are shown in Table 2. Last two layers are fully connected layers with a *relu* activation function. All the hyper-parameters are reported in Appendix II.

### 4.3 Results

Classifier's performance is evaluated based on micro accuracy and macro precision, recall and F-score. F-score is the combined metric of precision and recall, so accuracy and the F-score will be our main evaluation metric.

| Embedding | Classifier | A | P | R | F |
|---|---|---|---|---|---|
| baseline | ERF | 77.39 | 0.78 | **0.72** | **0.74** |
| | RF(N-gram) | 75.88 | **0.83** | 0.64 | 0.70 |
| Random | CNN+ | 66.15 | 0.63 | 0.57 | 0.59 |
| charcnn | CNN+ | 74.65 | 0.73 | 0.65 | 0.68 |
| bpe+w2v | CNN+ | 71.90 | 0.74 | 0.64 | 0.67 |
| bpe+glove | CNN+ | 74.13 | 0.76 | 0.64 | 0.68 |
| FastText* | MLP | 74.51 | 0.72 | 0.66 | 0.68 |
| FastText | LSTM+ | **77.68** | 0.76 | 0.71 | 0.73 |

Table 4: Evaluation measures are accuracy (A), macro precision (P), macro recall (R) and macro F-score (F). Results reported on the test data are averaged over 5 runs. '+' sign indicates end-to-end training integrated with classifier.

We have used two baseline models to compare against, first one is Krishna et al.'s (2016)

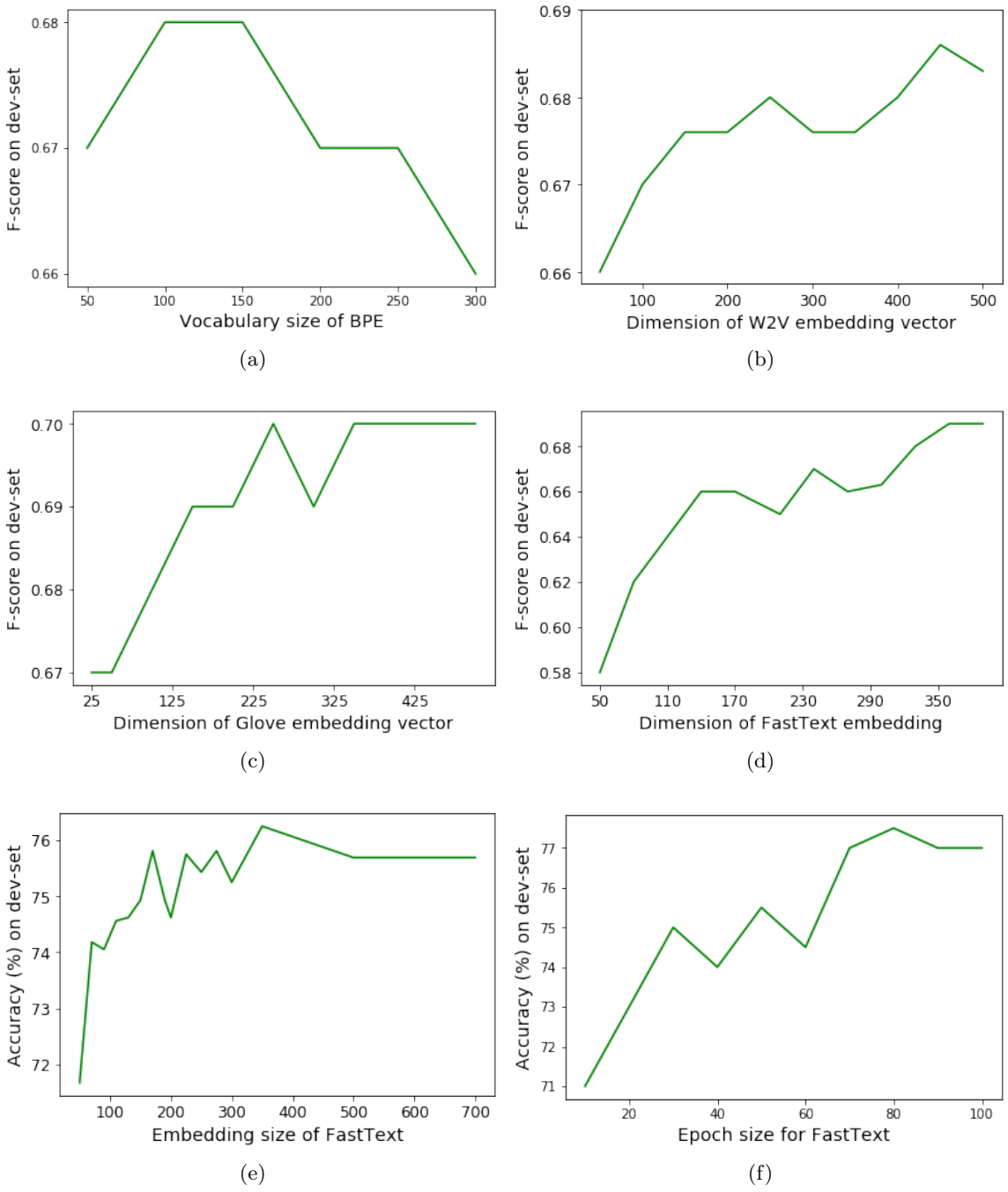
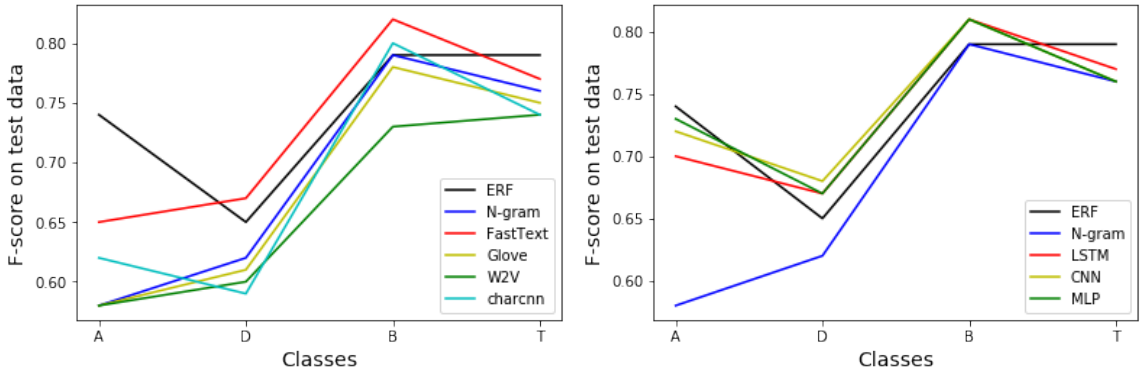



Figure 3: Investigating the sensitivity of the results (F1-score and Accuracy) with respect to the dimensionality of various embeddings on the development set: (a) As vocabulary size of BPE increases, macro F1-score decreases. So we have used the BPE vocabulary size as 100. (b) As embedding size of w2v increases, there is a gradual increase in F-score. So we have chosen 450 as the embedding size. (c) As embedding size of Glove increases, there is a gradual increase in F-score. So we have chosen 450 as the embedding size. (d) As the FastText dimension increases (with component-wise subtraction and product augmentation), there is a gradual increase in F-score. (e) Effect on accuracy as embedding size of FastText increases. For our experimentation, we have chosen embedding size of 350. (f) Effect on accuracy as epoch size varies for FastText.

feature engineered model with ERF classifier (F-score 0.74). Another baseline is N-gram based features with Random Forest (RF) classifier (F-score 0.70). In this model, only N-gram based




(a) (b)

Figure 4: (a) Class-wise F-score for different embeddings with the same architecture (CNN) (b) Class-wise F-score for different architectures with the same embedding (FastText). Note that class size increases as we move from left to right along x-axis. ERF and N-gram are baselines reported in Table 4.

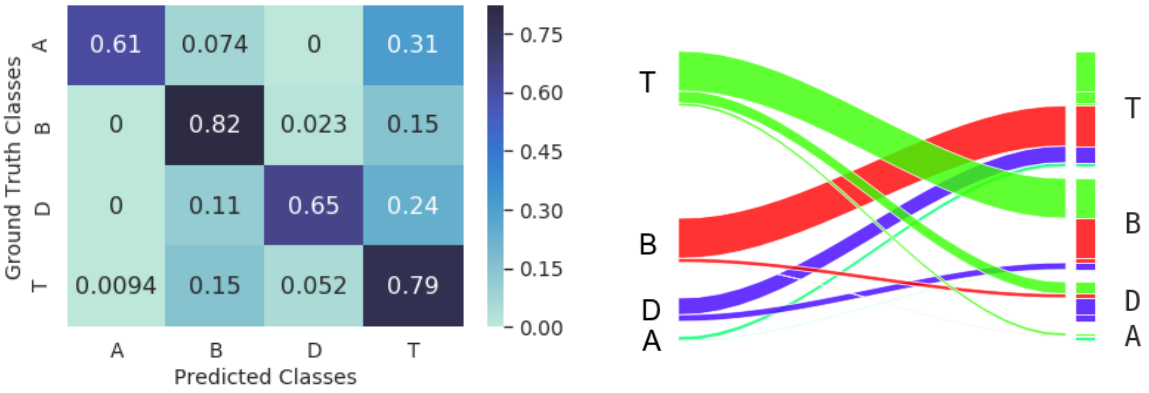
feature engineering is involved, but it was able to give comparable performance.

There are three possible ways to feed input to the system, namely, word level, subword level, and character level. Based on these categorizations, step by step, we evaluated our MLP, CNN, and LSTM based classifiers. First, for word-level inputs, we randomly initialized all the embedding vectors and checked the performance of the classifier. We were able to reach up to 0.59 (macro) F-score with CNN+ classifier (Table 4). Next, for subword level input, we used W2V and Glove embedding on BPE segmented (the segmentation is not morphemic) sub-words of the compound. These embeddings helped to get significant improvement compared to word level randomly initialized embedding, achieving F-score of 0.67 and 0.68, respectively. As shown in Figure 2(a), W2V and Glove could not give very good embeddings due to the rare occurrence of compound words in the corpus. Then we experimented with another embedding, FastText, which has shown excellent performance compared to all other systems. We were able to reach 0.73 (macro) F-score. We almost achieved state of the art result without feature engineering. Then we used the FastText* embedding combination technique to check whether we can improve further, but it declined the actual result to 0.68. Finally, character level input with CharCNN architecture with randomly initialized embedding reached 0.68. Our system outperformed in terms of accuracy (77.68) to state of the art baseline (77.39). We also integrated end-to-end training to learn task-specific embedding in all systems mentioned above. Detailed results for all the systems are presented in Appendix I.

### 4.4 Error Analysis

We have done a detailed analysis of particular instances of compound types which get misclassified. From confusion matrix heat map in Figure 5, we can see that most of the mis-classification has gone to *Tatpuruṣa* class for our best performing system. There are no mis-classification between *Dvandva* and *Avyayībhāva*. Specific sub-type of *Tatpuruṣa* has similar properties as that of *Avyayībhāva*, where first component of compound is *avyaya*, which creates conflict between these two classes. In our observation, 11 data-points from *Tatpuruṣa* got mis-classified into *Avyayībhāva* where all of them have the first component as *avyaya*. Also from Figure 5(a), we can see that most of the compounds from *Avyayībhāva* were misclassified into *Tatpuruṣa*. Our best model is able to perform better compared to the baseline model for *Bahuvrīhi* and *Dvandva* which are the second and the third most highly populated classes (Figure 4). Figure 5(b) indicates that our best system mostly got confused between *Tatpuruṣa* and *Bahuvrīhi*, because there is a special sub-type in both of these semantic classes which exhibits similar properties.

There are more than 600 unique components of compound common in training set of *Bahuvrīhi* and *Tatpuruṣa*. Out of these, 205 components have more number of occurrences in *Bahuvrīhi* than that of *Tatpuruṣa* and 201 components have more occurrence in *Tatpuruṣa* than that of *Bahuvrīhi*. So common component compounds present in a conflicting class which has less occurrences will be misclassified. Since we have not provided any other information, classifier is getting confused due to common component occurrences in both the classes. Similar cases have been found for *Dvandva* and *Tatpuruṣa*. For example, *bāla* occurred 7 times in *Dvandva* and 12 times in *Tatpuruṣa*, so majority of compounds of *Dvandva* having *bāla* as component will be misclassified into *Tatpuruṣa*. There are 11 such unique components in training set which have number of occurrences more than 4 in either class. We need to provide contextual information in order to overcome this problem. In summary, error cases observed in our best system are similar to that of baseline system. In this classification setup, apart from individual components of compounds, we have not provided contextual information or canonical paraphrasing. With this restriction, the classification problem is not entirely solvable; however, we explored up to what degree the ambiguities can be resolved.




(a) (b)

Figure 5: (a) Confusion matrix heat-map for our best performing system (A, B, D and T refer to *Avyayībhāva, Bahuvrīhi,Dvandva, and Tatpuruṣa,* respectively) (b) Alluvial graph for showing mis-classification to demonstrate conflicts between classes.

## 5 Related Work

Semantic analysis of compounds is an essential preprocessing step for improving on overall downstream NLP applications such as information extraction, question answering, machine translation, and many more (Fares et al., 2016). It has captivated much attention from the computational linguistics community, particularly on languages like English, Dutch, Italian, Afrikaans, and German (Verhoeven et al., 2014b). By rigorously studying Sanskrit compounding system and Sanskrit grammar, analysis of compounds in Hindi and Marathi has been done (Kulkarni et al., 2012). Another interesting approach uses simple statistics on how to automate segmentation and type identification of compounds (Kumar et al., 2010). Nastase et al. (2006) show that from two types of word meaning, namely, based on lexical resources and corpus-based, noun-modifier semantic relations can be learned. Another exciting work by Séaghdha and Copestake (2013) has done noun-noun compound classification using statistical learning framework of kernel methods, where the measure of similarity between compound components is determined using kernel function. Based on *Aṣṭādhyāyī* rules, Kulkarni and Kumar (2013) has developed rule-based compound type identifier. This study helped to get more insights on what kind of information should be incorporated into lexical databases to automate this analysis. Kulkarni and Kumar (2011) proposed a constituency parser for Sanskrit compounds to generate paraphrase of the compound which helps to understand the meaning of compounds better.

Recently, neural models are widely used for different downstream NLP applications for Sanskrit. The error corrections in Sanskrit OCR documents is done based on a neural network based approach (Adiga et al., 2018). Another work used neural models for post-OCR text correction for digitising texts in Romanised Sanskrit (Krishna et al., 2018a). Hellwig and Nehrdich (2018) proposed an approach for automating feature engineering required for the word segmentation task. Another neural-based approach for word segmentation based on seq2seq model architecture was proposed by Reddy et al. (2018), where they have shown significant improvement compared to the previous linguistically involved models. Feedforward networks are used for building Sanskrit character recognition system (Dineshkumar and Suganthi, 2015). Krishna et al. (2018c) proposed energy-based framework for jointly solving the word segmentation and morphological tagging tasks in Sanskrit. The pretrained word embeddings proposed by Mikolov (2013) and Pennington (2014) had a great impact in the field of Natural Language Processing (NLP). However, these token based embeddings were unable to generate embeddings for out-of-vocabulary (OOV) words. To overcome this shortcoming, subword level information was integrated into recent approaches, where character-n-gram features (Bojanowski et al., 2017) have shown good performance over the compositional function of individual characters (Wieting et al., 2015). Another interesting approach (Zhang et al., 2015) is the use of character level input for word-level predictions.

## 6 Conclusion

For resource-rich languages, deep learning based models have helped in improving the state of the art for most of the NLP tasks, and have now replaced the need for feature engineering with the choice of a good model architecture. In this work, we systematically investigated the following research question: Can the recent advances in neural network outperform traditional hand engineered feature based methods on the semantic level multi-class compound classification task for Sanskrit? We experimented with some of the basic architectures, namely, MLP, CNN, and LSTM, with input representation at the word, sub-word, and character level. The experiments suggest that the end-to-end trained LSTM architecture with FastText embedding gives an F-score of 0.73 compared to the state of the art baseline (0.74) which utilized a lot of domain specific features including lexical lists, grammar rules, etc. This is clearly an important result.

There are many limitations of this study. For instance, what is the effect of the corpus size on the performance? We work with a corpus with less than 5 million tokens, which is negligible compared to 840 billion tokens, on which Glove embeddings for English have been trained. Would a larger dataset have helped? Could methods based on cross-lingual embeddings help in this scenario for transfer learning from languages similar to Sanskrit?

## Acknowledgements

The first author would like to thank Pranav Kulkarni, IIT Kanpur, for his helpful feedback and suggestions.

## References

Devaraja Adiga, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K Ramasubramanian, and Malhar Kulkarni. 2018. Improving the learnability of classifiers for sanskrit ocr corrections. In *The 17th World Sanskrit Conference, Vancouver, Canada. IASS.*

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, Copenhagen, Denmark, September. Association for Computational Linguistics.

Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Antonietta Bisetto and Sergio Scalise. 2005. The classification of compounds. *Lingue e linguaggio*, 4(2):319–0.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September. Association for Computational Linguistics.

Silvio Cordeiro, Carlos Ramisch, Marco Idiart, and Aline Villavicencio. 2016. Predicting the compositionality of nominal compounds: Giving word embeddings a hard time. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1986–1997.

Corina Dima and Erhard Hinrichs. 2015. Automatic noun compound interpretation using deep neural networks and word embeddings. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 173–183.

R Dineshkumar and J Suganthi. 2015. Sanskrit character recognition system using neural network. *Indian Journal of Science and Technology*, 8(1):65.

Murhaf Fares, Stephan Oepen, and Erik Velldal. 2016. Identifying compounds : On the role of syntax.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning*, 63(1):3–42.

Brendan S Gillon. 2009. Tagging classical sanskrit compounds. In *International Sanskrit Computational Linguistics Symposium*, pages 98–105. Springer.

Roxana Girju, Dan Moldovan, Marta Tatu, and Daniel Antohe. 2005. On the semantics of noun compounds. *Computer speech & language*, 19(4):479–496.

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly learning word representations and composition functions using predicate-argument structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1544–1555.

Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Abhik Jana, Dima Puzyrev, Alexander Panchenko, Pawan Goyal, Chris Biemann, and Animesh Mukherjee. 2019. On the compositionality prediction of noun phrases using poincar embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence (Italy), July. Association for Computational Linguistics.

Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in neural information processing systems*, pages 641–648.

Su Nam Kim and Timothy Baldwin. 2005. Automatic interpretation of noun compounds using wordnet similarity. In *International Conference on Natural Language Processing*, pages 945–956. Springer.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October. Association for Computational Linguistics.

Amrith Krishna, Pavankumar Satuluri, Shubham Sharma, Apurv Kumar, and Pawan Goyal. 2016. Compound type identification in sanskrit: What roles do the corpus and grammar play? In *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*, pages 1–10.

Amrith Krishna, Bodhisattwa P. Majumder, Rajesh Bhat, and Pawan Goyal. 2018a. Upcycle your OCR: Reusing OCRs for post-OCR text correction in Romanised Sanskrit. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 345–355, Brussels, Belgium, October. Association for Computational Linguistics.

Amrith Krishna, Bodhisattwa Prasad Majumder, Anil Kumar Boga, and Pawan Goyal. 2018b. An ekalavyaapproach to learning context free grammar rules for sanskrit using adaptor grammar. *Computational Sanskrit & Digital Humanities*, page 83.

Amrith Krishna, Bishal Santra, Sasi Prasanth Bandaru, Gaurav Sahu, Vishnu Dutt Sharma, Pavankumar Satuluri, and Pawan Goyal. 2018c. Free as in free word order: An energy based model for word segmentation and morphological tagging in Sanskrit. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2550–2561, Brussels, Belgium, October-November. Association for Computational Linguistics.

Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for sanskrit compounds. *Proceedings of ICON*.

Amba Kulkarni and Anil Kumar. 2013. Clues from as. t. adhyayı for compound type identification. In *Proceedings of the International Sanskrit Computational Linguistics Symposium. DK Printworld (P) Ltd.*

Amba Kulkarni, Soma Paul, Malhar Kulkarni, Anil Kumar, and Nitesh Surtani. 2012. Semantic processing of compounds in indian languages. *Proceedings of COLING 2012*, pages 1489–1502.

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In *International Sanskrit Computational Linguistics Symposium*, pages 57–69. Springer.

John J Lowe. 2015. The syntax of sanskrit compounds. *Language*, 91(3):e71–e115.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Sivaja S Nair and Amba Kulkarni. 2010. The knowledge structure in amarakośa. In *International Sanskrit Computational Linguistics Symposium*, pages 173–189. Springer.

Vivi Nastase, Jelber Sayyad-Shirabad, Marina Sokolova, and Stan Szpakowicz. 2006. Learning noun-modifier semantic relations with corpus-based and wordnet-based features. In *AAAI*, pages 781–787.

Truc-Vien T Nguyen, Alessandro Moschitti, and Giuseppe Riccardi. 2009. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1378–1387. Association for Computational Linguistics.

Susan Olsen. 2000. Composition. *G. Booij and al*, pages 897–916.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Girishkumar Ponkiya, Pushpak Bhattacharyya, and Girish K. Palshikar. 2016. On why coarse class classification is bottleneck in noun compound interpretation. In *Proceedings of the 13th International Conference on Natural Language Processing*, pages 293–298, Varanasi, India, December. NLP Association of India.

Vikas Reddy, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M R, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan, May. European Languages Resources Association (ELRA).

Bryan Rink and Sanda Harabagiu. 2010. Utd: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 256–259. Association for Computational Linguistics.

Bahar Salehi, Paul Cook, and Timothy Baldwin. 2015. A word embedding approach to predicting the compositionality of multiword expressions. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 977–983.

Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. *arXiv preprint arXiv:1504.06580*.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.

Diarmuid O Séaghdha and Ann Copestake. 2013. Interpreting compound nouns with kernel methods. *Natural Language Engineering*, 19(3):331–356.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium, October-November. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, July. Association for Computational Linguistics.

Stephen Tratz and Eduard Hovy. 2010. A taxonomy, dataset, and classifier for automatic noun compound interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 678–687. Association for Computational Linguistics.

Gary Alan Tubb and Emery Robert Boose. 2007. *Scholastic Sanskrit: A handbook for students*. Columbia University Press.

Ben Verhoeven, Menno van Zaanen, Walter Daelemans, and Gerhard van Huyssteen. 2014a. Automatic compound processing: Compound splitting and semantic analysis for Afrikaans and Dutch. In *Proceedings of the First Workshop on Computational Approaches to Compound Analysis (ComAComA 2014)*, pages 20–30, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Ben Verhoeven, Menno van Zaanen, Walter Daelemans, and Gerhard Van Huyssteen. 2014b. Automatic compound processing: Compound splitting and semantic analysis for afrikaans and dutch. In *Proceedings of the First Workshop on Computational Approaches to Compound Analysis,(ComAComA 2014), Dublin, Ireland, August 24, 2014/Verhoeven, B.[edit.]; ea*, pages 20–30.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

## Appendix I

| Embedding | Classifier | A | P | R | F |
|---|---|---|---|---|---|
| baseline | ERF | 77.39 | 0.78 | **0.72** | **0.74** |
| | RF(N-gram) | 75.88 | **0.83** | 0.64 | 0.70 |
| Random | MLP | 64.9 | 0.62 | 0.55 | 0.58 |
| | MLP+ | 65.78 | 0.61 | 0.56 | 0.58 |
| | CNN | 65.91 | 0.60 | 0.55 | 0.58 |
| | CNN+ | 66.15 | 0.63 | 0.57 | 0.59 |
| | LSTM | 65.28 | 0.62 | 0.53 | 0.56 |
| | LSTM+ | 65.88 | 0.63 | 0.56 | 0.59 |
| charcnn | CNN | 74.32 | 0.72 | 0.65 | 0.67 |
| | CNN+ | 74.65 | 0.73 | 0.65 | 0.68 |
| bpe+w2v | MLP | 68.53 | 0.71 | 0.59 | 0.62 |
| | MLP+ | 69.85 | 0.74 | 0.58 | 0.63 |
| | CNN | 72.27 | 0.77 | 0.61 | 0.65 |
| | CNN+ | 71.90 | 0.74 | 0.64 | 0.67 |
| | LSTM | 67.48 | 0.71 | 0.60 | 0.63 |
| | LSTM+ | 68.94 | 0.73 | 0.60 | 0.64 |
| bpe+glove | MLP | 71.37 | 0.75 | 0.60 | 0.65 |
| | MLP+ | 72.12 | 0.73 | 0.63 | 0.66 |
| | CNN | 73.01 | 0.75 | 0.62 | 0.67 |
| | CNN+ | 74.13 | 0.76 | 0.64 | 0.68 |
| | LSTM | 69.17 | 0.72 | 0.60 | 0.63 |
| | LSTM+ | 69.42 | 0.71 | 0.62 | 0.64 |
| FastText* | MLP | 74.51 . | 0.72 | 0.66 | 0.68 |
| FastText | MLP | 76.77 | 0.75 | 0.71 | 0.72 |
| | MLP+ | 77.06 | 0.75 | 0.71 | 0.72 |
| | CNN | 77.04 | 0.76 | 0.71 | 0.73 |
| | CNN+ | 77.40 | 0.76 | 0.70 | 0.73 |
| | LSTM | 77.49 | 0.76 | 0.70 | 0.73 |
| | LSTM+ | **77.68** | 0.76 | 0.71 | 0.73 |

Table 5: Evaluation measures are accuracy (A), macro precision (P), macro recall (R) and macro F-score (F). Results reported on test data in table are averaged over 5 runs. '+' sign indicates end-to-end training integrated with classifier.

## Appendix II

| Embedding | Parameter | Description | Value |
|---|---|---|---|
| CharCNN | maxlen | maximum no of characters in input | 25 |
| | Voc-size | Vocabulary size of characters | 60 |
| | size | randomly initialized embedding size | 1014 |
| w2v | size | Dimensionality of the word vectors | 450 |
| | window | Max distance between current & predicted word | 15 |
| | BPE-Voc | BPE vocabulary size used for segmentation | 100 |
| | sample | down-sampling of more-frequent words | 1e-3 |
| | min-count | Ignores all words with frequency lower than this | 1 |
| | epochs | Number of iterations over the corpus. | 10 |
| Glove | size | Dimensionality of the word vectors | 450 |
| | window | Max distance between current & predicted word | 20 |
| | BPE-Voc | BPE vocabulary size used for segmentation | 100 |
| | min-count | Ignores all words with frequency lower than this | 1 |
| | epochs | Number of iterations over the corpus. | 70 |
| FastText | size | Dimensionality of the word vectors | 350 |
| | window | Max distance between current & predicted word | 11 |
| | min-n | Minimum length of char n-grams | 2 |
| | max-n | Maximum length of char n-grams | 11 |
| | epochs | Number of iterations over the corpus. | 70 |
| FastText* | input size | size of FastText features used as input | 350 |

Table 6: Hyper-parameters used in all the systems.